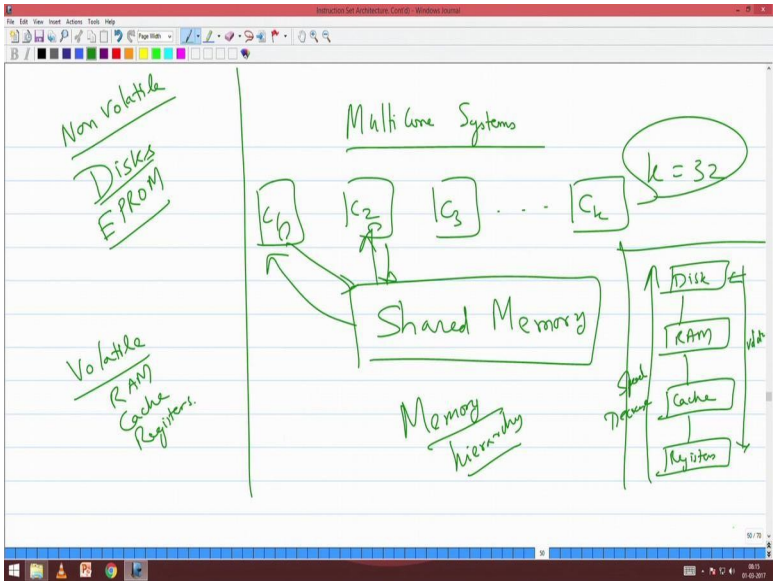
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 26
Lab 3: Virtual Memory

(Refer Slide Time: 00:22)



Good morning. We will now carry on with what we call as multi core systems. But as we proceed towards multi core system, what is important there? What is the difference between a single core system and a multi core system, from an organisation point of view or a from a architecture point of view? What is the new addition that we have going to get? When you look at a multi core system we will have several super scalar processes, several cores let me call it c 1, c 2, c 3 till c k. Today k can be as big as 32. Now there are several instead of one super scalar (Refer Time: 01:01) out of order branch predictor thing that we have seen we would have 32 such now what in addition? So, what should be learned in this multi core? The most important thing is, how all these things can work together. And the way they work

together is by passing actually messages between them.

One thing is we can do a traditional thing like you know I could have 32 different activities in your mobile phone right. So, you have 32 one, one mail server is running,

one mail demon is running, one whatsapp demon is running, one message SMS is running. So, each one can be put on to one, one core and it will be just running there. And whenever it is awakened, you get a message it will awakened it will do some activities.

So, like that several sequential applications, but instead of one CPU executing all the sequential applications I will just distribute this work load across these 32 processor or 8 processors. This is one easiest way of doing. So, what do you achieve by having a multi core processor, I have several task totally unrelated independent task like whatsapp and Gmail will talk to each other. But they are 2 independent task one instead of both being time shared on a same CPU, I can push it across different CPU and they can execute. So, this is how multi core are utilized today. There are very few really parallel applications. What is parallel? Suppose I want to add 100 numbers tell the program right. You will always say for i equal to 1 to 100 s is equal to s plus a, correct? That is a inherently sequential program. So, if I ask say if I have 4 processors take 25 and number each add them then finally, I add the answer that is the parallel way of looking it right. We have seen that in SIMD. This SIMD type of thinking is going to be very less or the number of code number of programs today that is running on your mobile phone which essentially exploit is the multi core facility there it is very, very less, you are getting this?

Because for several reasons people do not know how to write parallel program, people do not know and we do not have automatic parallelization environment though we are taking for years together. So, compilers may not have matured to that level right. So, there are lot of issues by which I cannot write parallel programs. So, the trivial thing is that just go and. So, what happens today if I have a quad. Core now 2 of the cores are like can never awake them up, they are sleeping eternally right. And some 2 cores will be used in which one core would be some 60 percent another will be 20 thirty percent right. The remaining core will be sleeping. So, because we do not know how to use it.

Now the best way of trying to use your mobile phone is develop a mobile app which will have some chess game with lot of artificial intelligence, which will start using these 4 cores do some a star search or some AI based searching and try to utilize. So, you have to write parallel programs to do it, I do not know whether android actually has an multi

programming or parallel programming environment at all, are you able to appreciate?

Now, the way So, the main issue here is that, if I have a multi core system I would like to if I want to exploit the speed of this multi core system, then I have to write parallel programs. I have to take one program split it into concurrent parts which are concurrently executable. Execute on these cores and get back the results aggregate the results and hence see the speed up. So, I should develop the acumen of writing parallel programs. So, towards the end of the course I will spend 2 or 3 classes basically to teach you how to write some parallel programs, very interesting parallel programs, simple parallel programs. I will also introduce a new model of computation you have already seen a model of computation in your data structure course right. The algebraic model of computation based on which you get your order notations. Now I will give you some you know parallel models of computation, I will also teach you that that makes this course very complete ok.

Now, in a right parallel programs the way one program will talk to another program is through passing messages. And the way they pass message there are many ways of passing messages we will look at those message passing system down, but one way by which you are current arm processors that you have on your chip and all process message is by between the cores is using the memory so that is why we call it as a shared memory architecture. I share lot of memory across these c 1 to c k, I share memory. And if I want one fellow if c 1 core 1 wants to send some result to core 2 how do they do they write into the memory, this fellow writes into the memory that fellow reads from this memory and vice verse, correct? So, we have to understand the memory management system in total before we start appreciating multi core systems. Because everything there centres around memory. We have learnt how to construct individual cores, we know how to construct them when I want to integrate these cores and give one model saying there are k cores use them and this is how you can use them and this is how you can analyse a program on them this is how you can program then before we even go into that aspect, it is very fundamental for us to understand how the memory is organised.

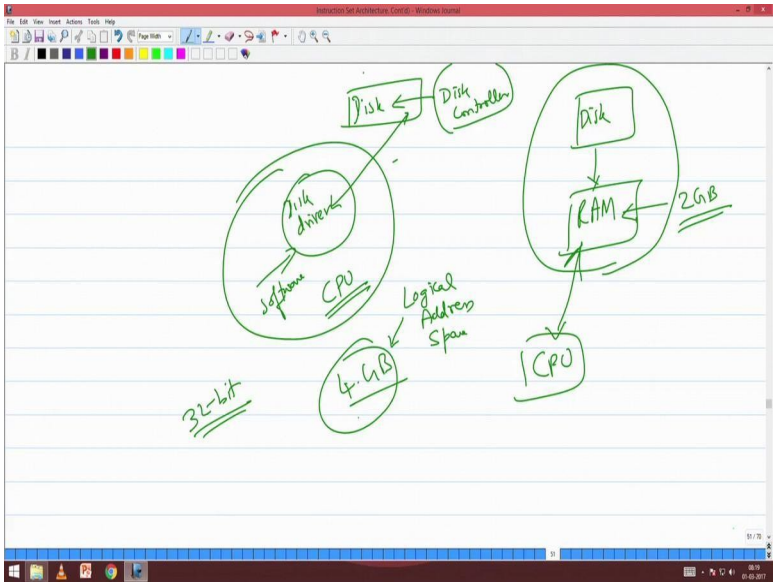
Right. So, in the next 3 to 4 classes it is also aligned to your assembly language assignment we will talk more about memory management, fair enough? And then I will

talk about how memory management is done on individual cores and then extend it to how that could be how that is extendable for multi core. We will we will do those 2 things before we go into programming these multi core processors ok. So, what are 2 types of memory? There are 2 types of memories one is a non volatile memory and another is a volatile memory. What do you mean by non volatile? When the power is switched off the memory contents do not get erased. A volatile memories when the power is switched off when current is not passed when electricity is not available when the power is not available, then what happens? Your memory gets erased. So, this is the difference between a non volatile memory and a volatile memory.

Now, let us say the non volatile memory that we see today are the disks plus something called erasable programmable read only memory, EPROM. These are all the non volatile memory. The volatile memory are your RAM your cache your registers. So, now, you have something called what we call as the memory hierarchy. What we understand by this term memory hierarchy, I have the disk from that I could load data into RAM, from RAM I can load data into caches, from cache I can load data to registers, or access directly. So, the volatility decreases. So, this is totally non volatile, while the remaining things are volatile, but the speed actually or speed actually, again decreases registers are faster than cache access. Caches are faster than RAM access RAM is much faster than disk access got this.

So, when I am looking at a single processor or single core whatever say c 1, I have to understand the memory hierarchy from it is perspective right. From c 1s perspective what is the memory hierarchy? Once I understand the perspective from c 1, c 1s perspective to the memory hierarchy then I can basically you know start you know elaborating on that and take it to the other stages. So, from an individual cores point of view, what is the memory hierarchy? We will

(Refer Slide Time: 10:11)



Now start looking at it. Let us come from the bottom. So, there is a disk right. And this fellow has a disk driver which is a software right. If you insert a u s b driver is being installed windows will give you that command right. This is fine and disk actually has a disk controller who understands how the disk works. This disk driver will be talking to the disk controller. Where will this disk driver execute? This will execute on your CPU, the disk controller will be a separate hardware and it will be, it will be executing. While this software will be used to program or talk to the disk controller, saying hey this is the data write it this is the data read it this is the data erase it etcetera. And the disk controller will do internally it will go and see whatever action which the you know, the OS wants it will get it done from it, ok.

So, this is running on the normal CPU, this is the software and that is the disk driver, which actually programs the disk controller to do lot of activity on this physical disks, correct? Now how do you read or write into a disk? From a disk I write it into a RAM. From the disk I write data to the RAM or programs to the RAM, and as a CPU I access only the RAM. I do not access the disk directly. Because the way instructions are executed it assumes that I have segmentation you have already done right. So, I have to load it into the memory and use that right. So, I the programming model that we have

conceived of only will look at RAM it will not look at disk.

So, you cannot execute a program directly from the disk, it actually takes copies that program into the RAM and executes from the RAM. Because that is very much necessary for all my intra security inter security, So many things that we have talked off from a segmentation point of view. Now let us look at what is happening at the RAM level. When I am writing a program I say I have a 32 bit architecture right; that means, I tell the programmer I tell the compiler you can address anywhere in this 32 bit. I am giving you a 4 GB address space, you can go and use this 4 GB address space. So, every program is given a 4 GB address space.

And it can go and use it, but ultimately this 4 GB is given this is called a logical address base, because physically it may not be present. Physically on your RAM I may have only 2 GB, but I am I as a programmer can access anywhere in this 4 GB. I will be given 4 GB logical address space, but actual RAM will be only 2 GB right. So, as for as the architecture goes with respect to the programmer, I go and say I am giving you 4 GB, irrespective of whatever I have so that 4 I may have 2 GB I can have 128 KB, I could have 1 MB whatever here physical memory, but I tell the user I tell the programmer hey I am giving you 4 GB. So, essentially; that means, I am telling the programmer here is a virtual memory, you assume that you have 4 GB that 4 GB is not physically present it is virtually present to you right.

So, I tell the programmer you write the program do not bother about anything about my physical RAM or anything. Just use assume you have that entire 32 bit address space of 4 GB write your program. I as an operating system with the support of the hardware will see that your 4 GB program will be executed with the small half a GB that I have. It is my responsibility to execute your 4 GB program on say a half GB that I have, are you able to follow?

So, essentially the architecture and the operating system together are presenting to the you know the user a 4 GB virtual address space. That is not physically present, why we call it as virtual? Because it is not physically present. It is not available in hardware. It is not available on the ram, but the OS and the architecture tells a user assume you do I will take care, are you appreciating this fact? That is why it is called a virtual memory because it is not physically available, all of you followed? Right.

Now, in which strength is it saying, suppose I go and say oh tomorrow morning I will go I will I will jump from a helicopter I will do sky diving right. There is no rationale behind that statement. I can not even climb 3 floors of ICSR. It is my head starts rounding, I go and say I will jump from helicopter tomorrow morning I will land directly in the fourth floor meaningless. So, there should be some rationale behind the operating system and the computer architecture trying to make such an assertive statement right. What is that rationale? Are you getting this? Right. So, what is the rationale? What could be that rationale? Rationale for making that statement, you were saying what will be the consequence of the statement.

Student: Entire program (Refer Time: 16:30) entire program is not executing at the same time.

That is the rationale.

So, can you make it? Entire 4 GB need not be every time 4 GB need not be there. So, can you can you make the rationale bit more affective?

Student: Only next instruction and data hold the next instruction.

Exactly that is the thing.

If I ensure that at any point of time, the next instruction to be executed and the data necessary for that next instruction to go to completion are available in the RAM. Then eventually my program will lead to completion. I do not need all the 4 GB at any point of time. At every point of time I need the next instruction to be executed in the RAM so that I fetch it I also need that data that is needed by the next instruction to go to completion, that also should be in the RAM, then only I can fetch the data. If this I ensure for every, every clock cycle of my entire program life then I can essentially go. And that next instruction can be assuming even it can be 1 KB next instruction, may not be not be thousand bytes, but even assuming it is 1 KB I can execute this entire program using including data I can finish off in 1 KB in assumptions, correct? Are you able to follow? So, so this is the rationale behind the virtual memory concept itself saying, I do not really

care how much you know underlying memory I have.

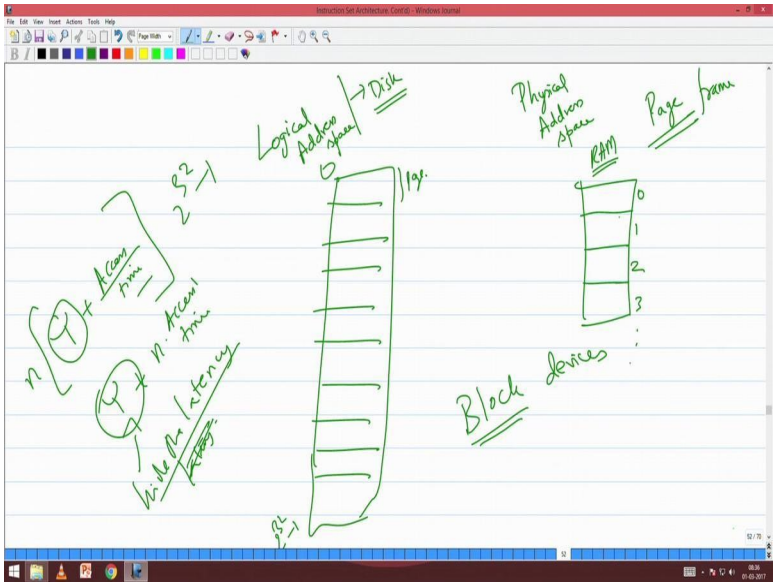
I should have something called RAM I cannot have 0 ram, but I will have something called ram, but if I have that something called RAM I can go and execute program of any size. So, as a 32 bit architecture you can only write programs that is as large as 4 GB, write that program with data everything come to me I will I will execute and irrespective of what I have in physical RAM. The rationale again is that always during the life time of a process at any point of time the next instruction to be executed and the data needed by the next instruction to take it to completion should be available. If this is ensured across the life of the program then essentially the program will go to completion. So, this is the rationale behind what we call as virtual memory, got it? Got it any doubts?

Student: (Refer Time: 18:58) instructions are executed in the pipeline right. So, like there might be situation like multiple instruction may need the data at the same time.

But that is all taken care by your dynamic scheduling. Load store, now you are talking of a load store architecture, somebody will 2 loads will come at a same point of time let them come. But your that that will be taken care by your dynamic scheduling, I will come to you what, I understand what you are talking of right. I will, I will come to you, there are multi port reads and so many things, we will, we will certainly look at these in great detail right.

First we will talk about virtual memory, what you are talking is about we will resolve it at a cache level, and there are some where going to be some structural hazard and so many things are going to come then we will have spit cache multi level cache. First we will finish this virtual memory right. Are you able to appreciate this [FL] now what happens is as follows.

(Refer Slide Time: 20:06).



So, how will I say. So, there is something called a logical address space and there is something called a physical address space. The logical address space is maintained in the disk, your OS course will teach you how it is maintained in a disk. Logical address space runs between 0 to in the 32 bit architecture it goes from 0 to 2 power 32 minus 1, all your segmentation everything is done on this logical address space.

So, if I want memory I have to give consecutive memory for a segment, all these things are done segment descriptor everything is done in the logical address space where entire segmentation is in the logical address space. Then what you do? Then there is some physical address space which is say one fourth of it is size for example, let me say I have one GB memory.

Now what happens is, what is my rationale? At every point of time the next instruction to be executed and the data needed by the next instruction should be loaded into memory, I will not load instruction by instruction. Why should I not load instruction by instruction? I will not load instruction by instruction because when I want to access memory what, how does the CPU access memory? Memory and all the peripherals are in a common bus. So, I have to go and request for the bus that bus fellow will bless me with the

request then I become the owner of the bus. And then I go on read memory and get it

back after that I relinquish that ownership and then somebody else will becomes the owner.

So, if I want to read from memory it is not a joke. It is a long drawn process right. You got it is a very long drawn process. So, I have to request the bus arbiter I have to get the control over the bus, then I have to send request to the memory then it will it will take it is own free time because it is much slower than the processor, then it basically sends the data back. I have to collect it at the CPU, then I have to do all these cache coherency all these you know policies, I have to collect it then I have to see all these policies I have to see whether there is a you know any segmentation violation all these things I need to check and then finally, it comes up right.

So, so every time I want to access data there is a latency involved right. So, if a CPU wants to access data there is a latency of tau involved, plus there is an access time. Access times is time to read one unit of data. So, this tau is contributed by the bus I asked the bus [FL] give me some control then the bus blesses me. That the time required to take the blessing of the bus will be this tau. After the tau only I will start accessing data. So, the total time will be tau plus access time of memory read or write access means read or write. So, if I want read to from I need some time that is given by this or write into memory let us take this. So, suppose I am doing n bits at a time one by one, this will be n into tau plus access time, but in one read itself I read n bits then it will be tau times n into access time hide, hide the latency.

Hide the latency because one time I am asking for the memory one time I am getting the request I immediately transfer n, n unit is of data right. Rather than one by one I. So, because of this reason this latency is quite significant it is not you know it is not something which is neglectable it is quite significant. So, if I multiply the tau with n then it goes back to my monkey example right. It becomes a drunkard monkey bit by a scorpion. So, whole thing becomes extremely complex. So, I would like to hide this latency and that is why when I want to transfer data from the a slower device to a faster devise whether it be from disk to RAM or RAM to cache I do not transfer one byte I transfer junk of bytes. So, these devises like a RAM your disk your cache they are all called block devices. What do you understand by the term block? Instead of reading one

byte at a time I read one block at a time.

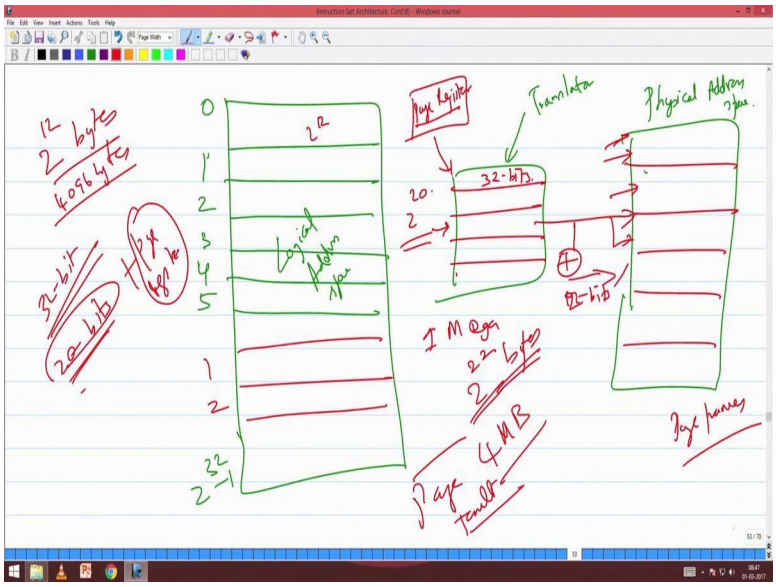
So, by basic granularity of trying to access these disks hard disk are basically a block. Where disk is also your hard disk is also a block device. So, when I read from the hard disk I do not read one byte, but essentially I read a set of bytes, which can be as long as or as large as 4 KB or 8 KB right. So, since you have now talking of block devices I will only move data in blocks, motivated by this I go and have this physical address space split into several blocks namely 0 1 2 3 etcetera. And each block from the virtual memory point of view, that is I am talking of the logical address space and physical address space, each block is called a page. Now in the physical address space this is the RAM what happens we split this RAM into equal parts each part is equal to the page size whatever page. So, similarly the logical address space will also be split into pages, each one is a page. So, your program and everything will be in the logical address space as and when required it will be moved from the logical address space to the physical address space for execution purpose is it right.

And the moment will not be as one byte after one byte it will be moved in terms of pages. So, I will make I will move if at all I moving something from a logical address space to the physical address space, the minimum granularity will be one page at least one page I have to move and vice verse. I cannot just move 1 byte or 2 byte etcetera. So, I will between the logical address space and the physical address space the moment will be in terms of one block of data which I call it as page. So, in the RAM the logical address space was split like this into equal parts.

The same equal part I can split the RAM also so that, a page from the logical address space can go and occupy this physical address space. So, the size of the page here in a logical address space is equal to the size of the page in the physical address space. So, these things like a photo sit is inside a photo frame right. If I have a photo frame. Why do you call it as a photo frame? Because photo will eventually I can keep a photo inside you can not go and sit inside that only a photo can sit. Similarly, a page sit is inside a page frame correct. So, all these things on the RAM are called page frames onto which some page from the logical address space currently stored in this can come and occupy. Do you get this? Are you able to appreciate this fact?

So, I have pages on my logical address space, I have page frame on my physical address space and anything that I move from the disk to the RAM it will be in the terms of blocks. In this case this block is called a page and I will be moving if at all a logical address space wants to talk to the physical address space, it is only thing that you can directly talk to. Then basically it is going to send pages of data. And those pages can come and sit in this page frames, are you able to follow?

(Refer Slide Time: 29:31)



Now So, how many? So, let us go to the next part, 0 to 2 power 32 minus 1 is the address space of the logical address space. In addition I will have something called a translator, which will basically translate, which will and then I will have something called cannot be as big as this is the physical address space. Now what will happen is, let me say this is page number one 0 1 2 3 like that it goes. Now what I do is that for every page I need to know, I need to translate for every page here. So how?

So, let us assume that each page is of size 2 power 12 bytes which is 4096. Each page is of size 3 power 12 bytes. And for every page I should know whether that page is still in the logical address space or it is moved into the physical address space. If it has moved into the physical address space I should know, which physical address space or which page frames. So, this will also have page frames. So, for every page in the logical

address space I need to know if at all it is loaded into the physical address space. If in

case it is loaded I should know which exactly is the page frame for which it is loaded are you able to follow? Yes or no. So, let us say. So, so how many entries should I have how many entries should I have.

Student: 2 power 20.

Um 2 power 20 why So much time for this? So, I have 2 power 12 each is 2 power 12 total is 2 power 32. So, I need to have 2 power 20 entries right. And each what is 2 power 20 is one mega bit right. One mega entries and what should this tell you? It should tell me which page frame it is loaded. So, so how many bytes I need for specifying the page frames? So, this is a 32 bit know I should tell which page frames is it is loaded. So, I may for every page I should know where which page frame and that page frame I may tell it as a beginning address of something address of these page frames. So, it will start right so that address would be again I need 32 bits, because I need to start this starting address of these page frames. Suppose I am storing like this. So, what is the total storage involved 4 bytes right. 4 bytes is 2 power 2. So, I need 2 power 22 bytes for this translation. That will be something like 4 MB 2 power 20 is 1 MB 4 MB. So, the size of this translated table itself will be 4 MB, correct? The size of the translated table itself will be 4 MB in which I will I by using that 4 MB what I can say? Whether a particular page is in the memory or not and if it is in the memory where in that memory this I can address by this right.

Now somebody should tell me where this table is? This table will be loaded into memory itself. Somebody should tell me where this table is which address it is available then only I can go and update these values correct. So, so I have some register let me call it as page register. This page register will tell me where this translating table starts. Now suppose I am generating a logical address which is 32 bit in size, I will use the first 20 bits to go into this table because this is this is 2 power 20 right. It will be 20 bits. So, I will use the first 20 bits to go to index into this table this. So, what I will do I will I will take the first 20 bits add it with page register and then go into this entry here, correct? In this entry what I will have is whether this page is loaded or I will have information whether this page is loaded or not. If this page is loaded then it will give me the start address of that page. To that start address I can add these 12 bits the least significant bits and that will

give me my actual address.

So, I say want to access a data. So, I am generating a 32 bit address that will be spawning across this logical address space, so that will be in some page number which is given by the first 20 bits because each page is 2 power 12, 12 bits 12 bit addressable like 2 power twelve. So, the first 20 bits will identify what that page is correct. So, I use the 20 bits as an index to this page register page register plus that 20 bit will give me the index. Where I can go and look at whether the page is already loaded into your physical address space, if it is loaded then I have to if it is loaded then I take the remaining 12 bits here add it to the you know this base address and I can get the actual address. If it is not loaded then I have to go and load it from the logical address space into the physical address space. If it is not loaded then the processor will create something called a page fault. When the processor creates a page fault because it is not available here, then I go and bring that page and load it into this memory and adjust that entry so that it points to it ok.

So, what we have seen today is a very simple set up where in I am having a logical address space which is 4 giga byte in size. I am asked I am having a translator and then I have a physical address space which is some very less value whenever. So, in the logical address space whenever a program generate a memory request right. That memory request essentially is send to the logical address space, it is generated from the logical address space. So, I get the 32 bit I take those 20 bits to index into table this is a translated table.

The starting entry of the translated table is stored in from page register I just add that many thing and index into this table. And this table will tell me whether the page I am looking for is available or not and if it is available, where it is available. If it is not available then the architecture will raise a page fault which will go and fetch that relevant portion from a physical memory from the logical address space in to this physical address space, you got a feeling? Now the biggest thorn in the flesh for this is, that I am wasting 4 mega bytes of data just for you know getting page.

Can I do something better? That is also there every time I am I am enabling paging in this scheme if this is the scheme that is final then what is the I need 4 MB of space just to

store page translation thing which is not acceptable. We can do something much better than this. So, this is the notion of virtual memory we will continue this in you know Mondays class, again on how these you know different aspects are taken care of. But this in essence this is the import of virtual memory right. And their translation, where in I go and say give me as much code you want I will execute it with as less memory I can have, right. We will further deliberate on these topics.